NOMBRE.....

NÚM. de MATRÍCULA..... GRUPO.....

Examen Final. Informática. Junio 2004

Instrucciones

- El examen consta de **diez** preguntas, cada una de las cuales puntuará **cero** o **un** punto.
- Se calificará con **un punto** si la respuesta correcta se indica en la forma **más simple**.
- La duración total del examen será de **dos horas**

---

1. Dada las siguientes secuencias binarias que corresponden a valores reales expresados en **coma flotante** (los 6 primeros bits corresponden a la mantisa y los cuatro últimos para el exponente, ambos en punto fijo y complemento a dos).
   a) 010000 0001 b) 001001 0010 c) 100001 0100 d) 011111 1111 e) 110000 1000
   ordenarlas de **menor** a **mayor** según el valor decimal representado. Nota: no es necesario indicar el valor representado. Es suficiente indicar la letra ordinal correspondiente al valor numérico binario.

2. Dado un tipo de dato `grupo` formado con elementos de tipo `persona`, completar la función `existeuno` para que devuelva **verdadero** (o **falso**, en caso contrario) si encuentra **al menos un** elemento en el parámetro `a` de tipo `grupo` que cumpla las **dos** condiciones siguientes: el **valor** del campo `dato1` debe ser igual o superior a **4 y** la **media aritmética** de los valores almacenados en los campos `dato1`, `dato2` y `dato3` debe ser igual o superior a **5**.

```
type persona=record
         nombre:string[20]; dato1, dato2, dato3: real end;
     grupo=array[1..100] of persona;
function existeuno(a:grupo):boolean;
```

```

```

```
    end;
```

3. Completar la función **recursiva** `dimpares` para que devuelva el **número** de **dígitos impares** del valor almacenado en el parámetro entero `n`. Por ejemplo, la llamada a `dimpares(0)` debe devolver el valor `0`, `dimpares(3)` devuelve el valor `1`, `dimpares(8)` devuelve el valor `0` y `dimpares(65304)` devuelve el valor `2`.

```
function dimpares(n:word):byte;
   begin
```

```

```

```
  end;
```

4. Completar la **unidad** `cadenas` para que incluya en su sección de **interfaz** un procedimiento llamado `inverso`. Este procedimiento debe **invertir** una cadena de caracteres almacenada en una variable de tipo `string`. Por ejemplo, si `a` es una variable de este tipo y almacena la cadena `'Hola'`, tras la ejecución de la llamada al procedimiento `inverso(a);`, la variable `a` debe tomar el valor `'aloH'`.

```
unit cadenas;
```

5. Construir la función `nlineas` para que devuelva el **número de líneas** de un archivo de texto que cumplan con la siguiente condición: el **último caracter** de la línea debe corresponder a un **dígito** decimal. El archivo de texto se ha asociado antes de la llamada a la función al parámetro de tipo `text`. Debe considerarse el caso de archivo vacío. Nota: debe considerarse que ninguna línea del archivo contiene más de 255 caracteres.

6. Completar el procedimiento `actualizarporcentaje` para que **añada** un **nuevo** valor al **final** de un archivo de números reales previamente asociado a la variable archivo `f`. El nuevo valor introducido es el resultado de **incrementar** el **último** valor almacenado en el archivo original en el **porcentaje** indicado en el parámetro `p`. Por ejemplo, si el último valor es `12.0` y `p` vale `50.0`, entonces el nuevo valor a añadir será `18.0`. Nota 1: Se debe considerar el caso de que el archivo esté vacío. En este caso en la ejecución del procedimiento debe almacenarse únicamente el valor `0.0`. Nota 2: La variable archivo **no** está previamente abierta.

```
type reales=file of real;
procedure actualizarporcentaje(var f:reales; p:real);
   var aux:real;
   begin
```

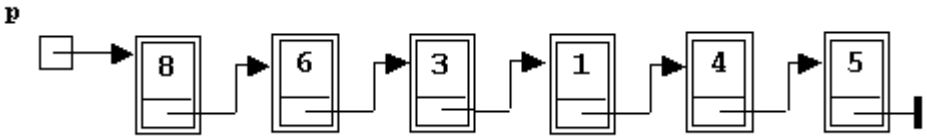
```
   end;
```

7. Construir un **procedimiento** `producto` para que, dados dos parámetros `m1` y `m2` de tipo `pmatriz`, proporcione o devuelva la **dirección** de memoria de una **nueva** variable dinámica de tipo `vector` que almacene en sus elementos los `n` valores resultantes de los correspondientes **productos escalares** de la **fila i**-ésima de la matriz a la que apunta `m1` por la **columna i**-ésima de la matriz a la que apunta `m2`.

```
const n=100;
type indice=1..n;
     vector=array[indice] of real;
     pvector =^vector;
     matriz=array[indice,indice] of real;
     pmatriz=^matriz;
```

8. Indicar las **declaraciones de tipo** necesarias para construir una estructura de tipo lista dinámica **doblemente** enlazada. En el campo `st` de los elementos de esta lista debe poder almacenarse la dirección de memoria del primer elemento correspondiente a otras estructuras de tipo lista dinámica **simple circular**. El campo `valor` de los elementos de estas listas circulares puede almacenar un dato de tipo `string`.

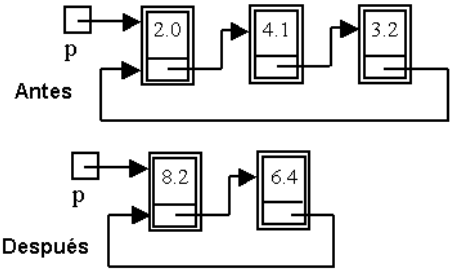
9. Completar la función **recursiva** `parimpar` para que devuelva el **número de veces** que, en una lista dinámica simple a cuyo primer elemento apunta `p`, a un valor **par** almacenado en campo `dato` de un elemento de la lista le sigue otro elemento con un valor **impar**. En el caso de lista vacía o si la lista tiene un único elemento la función debe devolver el valor `0`. En el caso de la lista de la figura debe devolver el valor `2`.




```
type puntero=^elemento;
     elemento=record dato:word; sig:puntero end;
function parimpar(p:puntero):word;
     begin
```

```
     end;
```

10. Completar el procedimiento `circular` para que modifique una lista dinámica **simple circular** según se muestra en la figura de la siguiente forma: debe **modificar** el valor del campo `dato` de **todos** los elementos de la lista multiplicándolos por el valor del campo `dato` del **primero** y, posteriormente, debe **eliminar** de la lista el **primer elemento** de la estructura (en consecuencia, es indiferente si ha modificado o no el valor del campo `dato` del primero). Originalmente, al primer elemento de la lista simple circular apunta el parámetro `p`. Si la lista está vacía no debe hacer nada. Si la lista tiene un único elemento sólo tiene que eliminarlo.




```
type ptr=^elemento;
     elemento=record
       dato:real;
       sig:ptr
       end;
procedure circular(var p:ptr);
```